# Lab: Day 1

**Media Computation**

## What you'll do:

Changing colors

Negatives

Grayscale

Making sunsets

## What you'll learn:

Jython Basics

Calling Functions

Assigning Variables

For loops

# Exercise 0: Getting Started

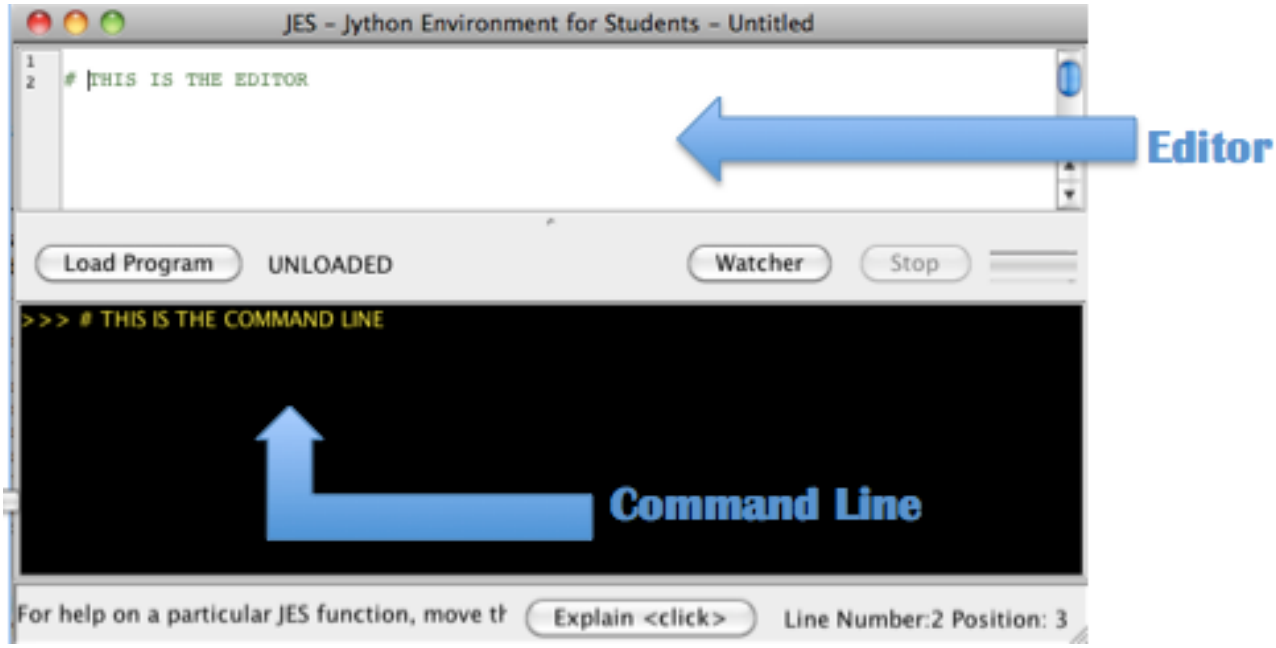
## Instructions:

- Go to *Places* – the drop-down menu in the upper left corner
- Go to *Home Folder*
- Go to *JES-4-3-nojava* *#this is the folder you will be working in*
- Create a **"lab1"** folder.

## Big Picture:

We want you to feel comfortable with the interpreter.

## Interpreter:




## Activity:

- Discuss the difference between the editor and the command line with your partner.
- Try to think of some benefits of each.
- Ask a lab assistant or teacher for verification of your ideas.

# Exercise 1: Jython Basics

**“**To succeed, you will soon learn, as I did, the importance of a solid foundation in the basics**”** -Alan Greenspan

## Command-line

Type each of the following expressions into the Jython prompt >>>, in the command-line, ending the line with the `Enter` key. Predict the results before you type them!!! Some of these expressions might cause Jython to error.

### Jython Expressions:

```
#this is a comment
3
2 + 3
5 + 6 + 7
-16 - -16
3 * 4 + 1
3 * (4 + 1)
from operator import add, mul
3 * 4
mul(3, 4)
mul(3, add(4, 1))
2 ** 3
pow(2, 3)
pow(pow(2, 3), abs(-2))
```

```
#do this column second
from math import sqrt, exp
exp(1)
sqrt(144)
pi
from math import pi
pi
pi * 3
print(pi)
print(4)
print(add(9,1))
print(print(2))
False or True
True and False
```

### Assigning Variables

```
x = 3
x
x + 1
x
x = x + 1
x
y = 1
y = y + 2
y
x = y
```

### Using the editor is next!

# Exercise 1: (cont)

## Command-line

Type each of the following expressions into the Jython prompt >>>, in the command-line, ending the line with the Enter key. Predict the results before you type them!!! Some of these expressions might cause Jython to error.

### Strings

```
“kit-kat bar”
“I am” + “adding strings”
var = “I am a saved string”
var[0]
var[10]
var[100]
```

### Lists

```
[1, 2, 3, 4]
[“I”, “can”, “contain”, “anything”]
[“different”, 1, 2, “data”, 8]
[1, 2] + [3, 4]
var = [1, 2, 3, 4]
var[0]
var[4]
var[5]
```

# Exercise 2: Defining your own functions

**Recall the structure of defining a function:**

def <name>(<arguments names>):
    return <expression>

## Command-Line

**Exercise 2(a):** At the python prompt >>>, type the following:

```
>>> def square(n):
...     return n*n
...
>>>
```

Be sure to indent the `return` statement correctly. Then, call the function `cube` with some numerical argument.

## Editor

**Exercise 2(b):** Now we will use the code editor to write a function. Rewrite the following into the editor.

```
def doStuff(x, y):
  return y + x * x
```

Now load the program into the Jython interpreter and test your function, by calling `doStuff` with two numerical arguments at the prompt.

**Exercise 2(c):** So you decide that you want `doStuff` to actually square y and add x (the opposite of what it is doing now). Edit the function in the editor, and test your function again to make sure it's doing the right thing.

# Exercise 3: PICTURES!

## **Exercise 3(a):** Collect the photos

### Instructions:

- Download pictures you want to work with and save them to your "lab1" folder.
  - Make sure all of your pictures are of medium size or smaller! **< 800x800 pixels**
  - Make sure all of your pictures are .jpg
  - When you Google images, you can filter by size on the left column

## **Exercise 3(b):** Manipulating Pictures

### Experiment:

- Try to remember what we went over in lecture in the first demo!
- Make a photo appear by typing into the editor and loading your program

  * You can look at what built-in functions we have by going to the "JES functions" drop-down menu

  *In the editor:*

  - make a file using your filepath* and assign that to a variable called `myFile`
  - make a picture out of the file and assign that to a variable called `myPic`
  - show that picture

  Load your program so that JES shows your picture!

  ** Your filepath should be equivalent to "lab1/filename.jpg"*

## **Exercise 3(c):** `pickAndShow`

- Define a function, pickAndShow—that does all of the above in one swoop—in your editor. It should take no arguments.
- Here's an example in pseudocode:

  def pickAndShow()

  - Has the user pick a file and assigns that to a variable
  - Turns that file into a picture and assigns that to variable
  - Shows that picture
- Once you have defined the function, call it inside of your editor
- Now save and load the program *# you should be able to choose a picture for it to show*

**Handy built-in functions that you *might* find helpful in 1(b) and 1(c):**

- pickAFile() – when called, allows the user to choose a file
- makePicture(someFile) – when called with some file, returns the picture form of the file
- show(somePicture) – displays the picture it is called with

## Exercise 3(d): `decreaseRed`

### Big Picture:

We want you to call the function `decreaseRed`, provided below, on a picture and display the result.

Note: We aren't having you write the function for yourself yet! We are just trying to get you comfortable with calling such functions!

Here is the code, type it into your editor:

```
def decreaseRed(picture):
      for p in getPixels(picture):
         value=getRed(p)
         setRed(p,value*0.5)
```

**Call this function on a picture with red in it, and show the result!**

### Things to keep in mind:

1. We need a picture to call this function on, so be sure to make one.
2. You want to write this all out inside of your editor.

## Experiment:

What if you decrease red again and again and again…? Try it!

# Exercise 3.5: Extras

Only do these if you have extra time. Or you can help other people. Or both!

## Recall the structure of defining a function:

def <name>(<arguments names>):
    return <expression>

**Attempt to write any of the following functions in python for practice**

1. **distance:** given two coordinates, which can be a list of two numbers (i.e. [1, 2]), attempts to find the distance between these two points.

```
def distance(point1, point2):
  your code here

>>> distance([1,0] , [1,3])
3
```

2. **Leap year:** given a year, determine if that year is a leap year.

Rules:

- If a year is divisible by 400 then it is a leap year.
- If a year is divisible by 100 and not by 400, then it is **not** a leap year.
- If a year is divisible by 4 then it is a leap year
- Otherwise, it is **not** a leap year.

```
def leapyear?(year):
  your code here

>>> leapyear?(1987)
3
```

# Exercise 4: Changing Colors

Recall the structure of a for loop…

```
for item in items:
    do something with item          # indentation!!!
```

## Exercise 4(a): `increaseRed`

Write the function, `increaseRed`, which should take in a picture and increase the red in the picture. Did anything interesting happen?

Here's an example in pseudocode:

```
def increaseRed(somePicture):
    for each pixel in the picture
        - get the red value and assign it to a variable
            - set the red value to be slightly higher by multiplying by some constant
    show the updated picture
```

## Handy built-in functions that you'll find helpful:

* You can find these by looking at the "JES Functions" drop-down menu in the bar at the top of your computer. They can be found under the functions for pictures and pixels!

- getPixels(picture)
- makeColor(redValue, greenValue, blueValue)
- getColor(pixel)
- setColor(pixel, newColor)
- getRed(pixel)
- setRed(pixel, redness)

## Exercise 4(b): `clearBlue`

Here's an example in pseudocode:

```
def clearBlue(somePicture):
    for each pixel in the picture
        - set the value of blue to 0        # why don't we need to save the original blue?
    show the updated picture
```

# Exercise 5: `negative`

**Strategy:**

- Let's think it through
  - R, G, B go from 0 to 255
  - Let's say Red is 10. That's very light red.
    - What's the opposite? LOTS of Red!
  - The negative of that would be 245: 255-10
- So, for each pixel, if we negate each color component in creating a new color, we negate the whole picture.

**Exercise 5(a):** `negative`

- Define a new function named `negative` that takes a picture as input and makes that picture negative.

(Go to the next page if you are stuck)

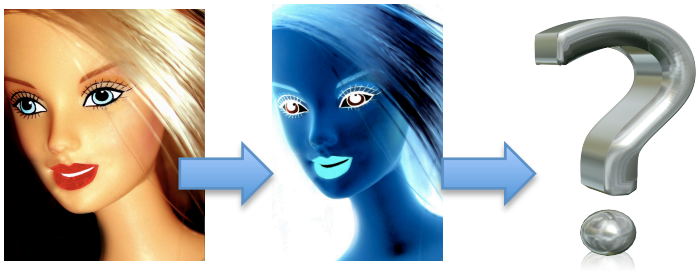
## Exercise 5(a): `negative`

Here's an example in pseudocode:

```
def negative(somePicture):
    for each pixel in the picture
        - find the red, green and blue values of the pixel and save those values to variables
        - make a new color with the new values being 255 – the old values
        - set the color of the picture to be the new color
    show the updated picture
```

Handy built-in functions that you'll find helpful:

```
getPixels(picture)
makeColor(redValue, greenValue, blueValue)
getColor(pixel)
setColor(pixel, newColor)
getRed(pixel)
getGreen(pixel)
getBlue(pixel)
setRed(pixel, redness)
setBlue(pixel, blueness)
setGreen(pixel, greenness)
```



## Exercise 5(b): double negative

"A double negative occurs when two forms of a negation are used in the same clause...canceling one another and producing an affirmative sense" ~Wikipedia

No! Not *that* "double negative!" That's what you get for quoting such a reliable source. ☺

### Concept:

Actually, the idea that a double negative results in a positive isn't that far off from the purpose of this exercise. Calling negative once should give us the opposite picture. If we call it again, should it not give us the original picture?

### Experiment:

Try calling the function twice. Once to get the negative of our original picture, and then again on that negative to get the original back! Does it work?

# Exercise 6: `grayscale`

"When you photograph people in color you photograph their clothes. But when you photograph people in B&W, you photograph their souls!" ~Ted Grant

## The Big Picture:

- We know that if red=green=blue, we get gray
  - But what value do we set all three to?

- What we need is a value representing the darkness of the color, the *luminance!*

- There are many ways, but one way that works reasonably well is dirt simple—simply take the average:

$$\frac{(red+green+blue)}{3}$$

## Exercise 6(a): `grayscale`

- Define a new function named `grayscale` that takes a picture as input and converts that picture to black and white!

(use the next page for reference if you are stuck)

## Exercise 6(a) (cont): `grayscale`

Here's an example in pseudocode:

```
def grayscale(somePicture):
    for each pixel in the picture
        - find the sum of the red, blue and green values of that pixel and assign
            that to a variable
        - divide that by three and assign that to some variable representing the
intensity
        - set the color of the picture to be the new color
    show the updated picture
```

Recall the structure of a for loop…

```
for item in items:
    do something with item
```

Handy built-in functions that you'll find helpful:

```
getPixels(picture)
makeColor(redValue, greenValue, blueValue)
getColor(pixel)
setColor(pixel, newColor)
getRed(pixel)
getGreen(pixel)
getBlue(pixel)
setRed(pixel, redness)
setBlue(pixel, blueness)
setGreen(pixel, greenness)
```

- After you've fully created your function, load your program and call this function at the bottom of your program by typing:

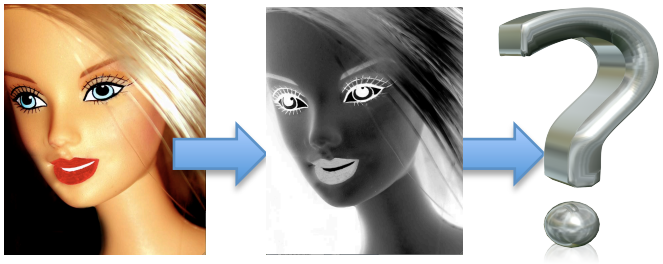
```
> grayscale(somePicture)
```

- Then show that picture!

```
> show(somePicture)
```

You should now see your picture in black and white!

## Exercise 6(b): Double grayscale



Concept:

With our negative function, we were able to get our original picture back by calling the function twice!

Experiment:

Try this with grayscale! What happens? Try to explain why to your partner!

## Exercise 6(c): Building a better grayscale!

Concept:

In reality, we don't perceive red, green, and blue as *equal* in their amount of luminance: How bright (or non-bright) something is.

- We tend to see blue as "darker" and red as "brighter"
- Even if, physically, the same amount of light is coming off of each

Experiment:

Rewrite grayscale to set each level (red, green, blue) to the getRed() of the pixel. Compare this to the result of setting each level to the getBlue() of the pixel. How do they compare to the original grayscale? Make a guess before you try it!

We will show you our version of grayscale towards the end of lab, but try for yourself!

# Exercise 7: `makeSunset`

We are now going to combine some of the ideas we have practiced so far, by taking a normal beach picture and generating a new picture that looks like a sunset.

Strategy: A sunset picture is more red.

- Find a picture online with some normal beach (not already with a sunset). Save the photo into your [Name] folder.
- Define a new function named `makeSunset` that takes a picture as input.
- **The Big Picture:** We want to use the idea of looping through the pixels in a picture (just like the previous exercises) to create a photo that looks more like a sunset. A sunset has more red coloring!

- Question: Does our sunset function generate a believable sunset image when we increase the redness by a certain factor? Try this:

Alternative Strategy: As the sun sets, less blue and green is visible, which makes things look more red.